REAL-TIME SCHEDULING FOR DEPENDABLE MULTIMEDIA TASKS IN MULTIPROCESSOR SYSTEMS

Xiao Qin Liping Pang Zongfen Han Shengli Li
Department of Computer Science, Huazhong University of Science and Technology
Wuhan, P.R. China, Post Code 430074
e-mail : xqin@371.net

Abstract: This paper proposes real-time scheduling algorithm for dependable multimedia tasks. The dependable multimedia tasks are executing on an integrated platform of a multiprocessor system. The algorithm can schedule dependable multimedia tasks with traditional real-time tasks. In order to meet the requirements of fault-tolerance, we apply primary/backup approach. When an instance of the multimedia task arrives in the system, the scheduling algorithm first multiplexes the primary instance into the multimedia server, then schedules the backup instance to another processor as late as possible. Compared to using a spare processor as a backup processor to be used in the case of a failure, our algorithm achieves high schedulability for both dependable multimedia tasks and traditional real-time tasks.

Keyworks
Multimedia server, dependable, fault-tolerant, real-time task, multiprocessor systems.

I. INTRODUCTION

Multimedia applications are soft real-time applications that have real-time requirements such as delay. There are two kinds of multimedia applications. The first type has no fault-tolerance requirement. Real-time requirements for this kind of multimedia are not as stringent as those hard real-time applications, and missing deadline of the multimedia task does not mean a significant failure. Another kind of multimedia task is dependable which has fault-tolerance requirement. The scheduling goal for such multimedia tasks are twofold: timeless (real-time) and dependability (fault-tolerance). Multimedia applications in space exploration, plan control, robotics control, and aircraft avionics are traditional dependable multimedia applications.

A number of approaches have investigated to cope with real-time scheduling for multimedia tasks. Paper [1] has studied rate monotonic and earliest deadline first scheduling algorithms for multimedia research systems and test beds. The drawback of rate monotonic algorithm is that it only deals with periodic tasks and could not schedule dynamically. When system is overloaded, earliest deadline first scheduling becomes non-deterministic. Paper [2] proposes a mechanism to provide the multimedia applications and traditional hard real-time applications using Spring system [3]. CPU scheduling and IPC mechanisms for operating systems for continuous media applications are investigated in [4]. Unfortunately, these real-time scheduling algorithms for multimedia tasks assume that the processors and the multimedia tasks are fault-free. The failure of the processor, on which some dependable multimedia tasks are running, may give rise to some destruction of equipment or even threat to human life.

A great deal of effort has been invested to address the joint requirement of real-time and fault-tolerance [5]. In real-time multiprocessor systems, dependability can be provided by scheduling multiple copies of tasks on different processors 6,7,8]. Primary/backup copies (PB) and triple modular redundancy (TMR) are two basic methods that allow multiple copies of a task to be scheduled on different processors [9]. Example of PB scheme can be found in [10]. We have explored algorithms using PB approach to schedule dependable real-time tasks with non-dependable real-time task [11, 12]. Balaji has studied an algorithm to dynamically distribute the workload of a failed processor to other operable processor [13]. Such algorithm assumes that the task set is fixed and the system has the enough processing power to execute tasks. Imprecise computation model presents timing faults by reducing the accuracy of results in a graceful manner [14,15,16]. However, these fault-tolerant techniques in real-time system are not suitable for real-time multimedia tasks.

In this paper, we concentrate on fault-tolerant scheduling algorithm for multimedia tasks. Our algorithm will employ the primary/backup approach that allows two copies of a multimedia task to be scheduled on two different processors in multiprocessor system. The primary copy of the multimedia task will be scheduled into a multimedia server that has been studied in [2]. The remainder of the paper is organized as follows. In section 2 we describe the system assumptions, and the system model is given in session 3 formally. In section 4, the scheduling algorithm is proposed. In section 5, we conclude the paper with a discussion of possible extension.

II. SYSTEM ASSUMPTIONS

We introduce a fault-tolerant scheduling approach for multimedia tasks in multiprocessor system. The system is considered to be a N interconnected identical processors and we assume that there is a task scheduling processor which maintains a global schedule. The scheduling algorithm studied in this paper also can be used in the distributed schedulers. In some multiprocessor system, some processors are allocated to process multimedia applications and other processors are dedicated to real-time applications. Each processor in such multiprocessor system has specific function, but in a dynamic environment, such system does not perform efficiently. If each processor can process both multimedia tasks and traditional real-time tasks, the system will become more flexible.

In fact, many real-time applications integrated real-time tasks and multimedia tasks. For example, in an Intensive Care Unit of the hospital, monitor gets video streams through cameras and real-time control actuators based on the video information. Our multiprocessor system is an integrated platform that can meet the requirements of both multimedia tasks and traditional real-time tasks.

Before giving the system assumption, we first describe our scheduling strategy. The scheduler will dynamically generate schedules in which every task in the system is guaranteed its required resources. If the primary copy of a multimedia task can not be guaranteed to complete before its deadline, this task will be rejected. If the completion guarantee can be assured for the primary copy of a multimedia task, but the timing constraint of its backup copy will not be guaranteed, there are two approaches to deal with this situation. One way is to reject this multimedia task (both reject its primary and backup copies). Another method is only rejecting the backup copy of this task, admitting to execute its primary copy and notifying the task that the fault-tolerant requirement of this task will not be met. That is to say, if the processor fails, this dependable multimedia task has no ability of continuing to complete before its deadline.

The assumptions of the system are described as follows:
Assumption 1: The tasks are assumed to be independent and non-preemptive.

Assumption 2: Each multimedia task in the system is dependable and has primary copy and a backup copy. Each dependable multimedia task only has one backup copy.

Assumption 3: Multimedia tasks on the multiprocessor system are guaranteed to complete if a processor fails at any instant of time and if a second processor does not fail before the system recovers.

Assumption 4: Time used for fault detection is assumed to be zero, that is the fault detection mechanism can immediately detect a fault without any delay.

III. SYSTEM MODEL

Each processor has a multimedia server, which is used to execute the multimedia tasks. The server is allocated a fraction of CPU time [2]. Task executions of more than one multimedia task are multiplexed into one multimedia server instance. The multimedia server can be regarded as a real-time task; it will be scheduled with other traditional real-time tasks in the system.

We will use the following notations concerning the set of multimedia servers in the multiprocessor system model.

- N is the total number of processors in the multiprocessor system.
- P is the set of N processors in the multiprocessor system, that is, $P = \{P_1, P_2, ..., P_N\}$
- MS is the set of N multimedia server in the multiprocessor system, each process has one multimedia server to control executions of multimedia tasks, that is, $MS = \{MS_1, MS_2, ..., MS_N\}$
- m(i, j) and mi(i, j) denote a multimedia task and its instance respectively, m(i, j) and mi(i, j) are tuples, that is, m(i, j) = mi(i, j) = (a, s, c, p, b), where a, s, c and p are arrival time, start time, computation time and period of m(i, j) respectively, b indicates on which processor its back copy instance is assigned. If the deadline of task m(i, j) is guaranteed, we have $a(i, j) \le s(i, j) \le a(i, j) + p(i, j)$ c(i, j).
- M_i M(i, j) are the total number of multimedia tasks in multimedia server MS_i and server instance MI(i, j) respectively.
- MS_i is a multimedia server, where $1 \le i \le N$. $MS_i = \{m(i, 1), ..., m(i, j), ..., m(i, M_i)\}$.
- MI(i, j) denotes the server instance for multimedia server MS_i; where $1 \le i \le N$. MI(i, j) = (s, c, p, Ω), where s, c, and p represent start time, computation time and period, Ω is the set of multimedia task instance.
- t(i, j) is a traditional real-time task in processor P_i, t(i, j) is a tuple, that is, t(i, j) = (a, s, c, p), where a, s, c and p are arrival time, start time, computation time and period of t(i, j) respectively.
- T_i is the total number of real-time tasks in processor P_i.
- RT_i is a set of real-time tasks in processor P_i, where $1 \le i \le N$. $RT_i = \{t(i, 1), ..., t(i, j), ..., t(i, T_i)\}$.

IV. SCHEDULING ALGORITHM

There are two main parts of the scheduling algorithm to schedule a multimedia task instance. The first part is to

multiplex the primary copy instance of the multimedia task into a multimedia server; the algorithm assigns the primary copy instance individually to a server instance. Part two focuses on how to allocate the backup copy instance of this task into another proper processors.

When scheduling the primary copy instance, algorithm tries to allocate it to the nearest server instance. Each primary copy instance must be assigned to complete before its deadline, if such server instance is not found, a new server instance must be generated. The server instance knows how many primary copy instances multiplexed into this server instance and in what order to execute these instances. Algorithm *Sch_Primary* is the designed to schedule the primary copies of the multimedia tasks.

Let r be the scheduling result, if the primary copy can not be guaranteed to complete before the deadline, and is rejected by the algorithm, r equals to 1. If r = 0, it means the primary copy instance is admitted and multiplexed to the server instance.

The scheduling algorithm for primary copy instance is described as follows:

ALGORITHM Sch_Primary

Input: Primary copy instance mi(i, j);
Output: r
IF (There is no server instance in the processor P_i) {
IF (Time interval [x, y] can be found, which satisfy:
y x = m(i, j).c and y ≤ mi(i, j).a + mi(i, j).p and
~((1≤ α ≤ T_i)∃α(x < t(i, α).s < y))) {
Generate a new multimedia server instance MI_i' ;
MI_i' .s = x; MI_i' .c = mi(i, j).c; mi(i, j).s = MI_i' .s; MI_i' .Ω = {mi(i, j)}; exit(0);
}
ELSE reject mi(i, j); exit(1);
}

/*
* There are some server instances, let MI(i, k)
* be the nearest one
*/
IF (mi(i, j).a + mi(i, j).c ≤ MI(i, k).s) {
IF (MI(i, k).s ≤ mi(i, j).a + mi(i, j).p) {
IF (~((1≤ α ≤ T_i)∃α(MI(i, k).s mi(i, j).c
< t(i, α).s < MI(i, k).s))) {
/*
*Multiplex primary copy instance in the head
* of the multimedia server instance M_i
*/
mi(i, j).s = MI(i, k).s mi(i, j).c;
MI(i, k).s = mi(i, j).s;
MI(i, k).c = MI(i, k).c+ mi(i, j).c;
MI(i, k).Ω = MI(i, k).Ω + {mi(i, j)};
exit(0);
}
IF (MI(i, k).s + MI(i, k).c + mi(i, j).c ≤
mi(i, j).a + mi(i, j).p and
~((1≤ α ≤ T_i)∃α(MI(i, k).s + MI(i, k).c <
t(i, α).s < MI(i, k).s + MI(i, k).c + mi(i, j).c))) {
/* Multiplex primary copy instance in the tail
* of the multimedia server instance M_i
*/
mi(i, j).s = MI(i, k).s + MI(i, k).c;
MI(i, k).c = MI(i, k).c + mi(i, j).c;
MI(i, k).Ω = MI(i, k).Ω + {mi(i, j)};
exit(0);
}
ELSE {
reject mi(i, j);
exit(1);
}
}
IF (Time interval [x, y] can be found, which satisfy:
y x = mi(i, j).c and y ≤ m(i, j).a + mi(i, j).p and
~((1≤ α ≤ T_i)∃α(x < t(i, α).s < y))) {
Generate a new multimedia server instance MI_i' ;
MI_i' .s = x;
MI_i' .c = mi(i, j).c;
mi(i, j).s = MI_i' .s;
MI_i' .Ω = mi(i, j);
exit(0);
}
ELSE {
reject mi(i, j);
exit(1);
}
}

IF (MI(i, k).s + MI(i, k).c + mi(i, j).c ≤ mi(i, j).a + mi(i, j).p and ~((1≤ α ≤ T_i)∃α(MI(i, k).s + MI(i, k).c< t(i, α).s < MI(i, k).s + MI(i, k).c + mi(i, j).c))) {
/*
* Multiplex primary copy instance in the tail
* of the multimedia server instance M_i
*/
mi(i, j).s = MI(i, k).s + MI(i, k).c;
MI(i, k).c = MI(i, k).c + mi(i, j).c;
MI(i, k).Ω = MI(i, k).Ω + {mi(i, j)};
exit(0);
}
ELSE {
reject mi(i, j);
exit(1);
}

In order to make utilization of the processor more efficient, and obtain high schedulability for the system while support

fault-tolerance for multimedia tasks, we use three methods. (1) Backup copies of multiple primary tasks are allowed to be scheduled in the same time period.
(2) If the primary copy instance of the multimedia task complete successfully, its backup instance will be deal located immediately.
(3) Schedule backup instance as late as possible.

Let r be the scheduling result. If the algorithm does not find a proper process for the backup copy, such backup copy will be rejected, and the scheduling algorithm will notify the task that the fault-tolerant requirement of this multimedia task can not be met, r also will be set to 1. If the backup copy instance can be scheduled to complete before its deadline, r will be set to 0.

Before presenting scheduling algorithm for backup copy instance, we described some useful notations as follows.

- mb(i, j) denotes the backup instance of primary multimedia task instance mi(i, j).
- Let π_β be free slot start time, that is [π_β, π_β+ mb(i, j).c] is a free slot in processor P_β in which no traditional real-time tasks and no server instance is scheduled to execute in this interval, formally speaking,
  $\sim((1 \le \alpha \le T_\beta) \exists \alpha(\pi_\beta < t(i, \alpha).s < \pi_\beta + mb(i, j).c))$
  and $\sim(\exists \alpha(\pi_\beta < MI(\beta, \alpha).s < \pi_\beta + mb(i, j).c))$.
- $\Pi_k(mb(i, j))$ is the set of free slot start time π, $\Pi_k(mb(i, j)) = \{\pi \mid [\pi, \pi + mb(i, j).c]$ is a free slot for backup instance mb(i, j) in processor $P_k\}$.
- $\Pi_{max}(mb(i, j))$ is the set of free slot start time, which defined as: $\Pi_{max}(mb(i, j)) = \{\pi_k \mid \pi_k = max(\Pi_k(mb(i, j)))\}$
- B_i is the total number of backup copy instances in processor P_i, where $1 \le i \le N$.
- MB_i is a set of backup copy instances in processor P_i, where $1 \le i \le N$. $MB_i = \{mb(i_1, 1), ..., mb(i_2, j), ..., mb(i_{Bi}, B_i)\}$, where $i_k \ne i$, $1 \le k \le B_i$.

ALGORITHM Sch_Backup

```
Input: Backup copy instance mb(i, j);
Output: r
/* Calculate set Π_max (mb(i, j)) first */
Π_max (mb(i, j)) = ∅;
/*
* Find the proper slot time for mb(i, j) on other processors
*/
For (k = 1; k ≤ N; k++) {
        IF (k ≠ i)
        /*
         * Do not schedule mb(i, j) on processor i
         * on which m(i, j) is assigned
         */
            IF (Π_k(mb(i, j)) ≠ ∅)
                Π_max (mb(i, j)) =
                        Π_max (mb(i, j)) + max(Π_k(mb(i, j)));
}
/* There are no free slot time for backup copy instance */
IF (Π_max (mb(i, j)) = ∅) {
        Reject mb(i, j);
        Notify the task that the fault-tolerant requirement
        of m(i, j) can not be met;
        exit(1);
}

/*
* Find the proper free slot,
* and schedule mb(i, j) as late as possible
*/
π_k = max(Π_max(mb(i, j)));
mb(i, j).s = π_k;
m(i, j).b = k;
MB_i = MB_i + {mb(i, j)};
B_i = B_i + 1; /* Insert mb(i, j) into set MB_i */
exit(0);
```

V. CONCLUSIONS

The dependable multimedia tasks are executing on an integrated platform of a multiprocessor system. The algorithm can schedule dependable multimedia tasks with traditional real-time tasks. In order to meet the requirements of fault-tolerance, we apply primary/backup approach. When a instance of the multimedia task arrives in the system, the scheduling algorithm first multiplexes the primary instance into the multimedia server, then finds a proper processor to schedule the backup instance as late as possible. Our algorithm achieves high schedulability for both dependable multimedia tasks and traditional real-time tasks compared to using a spare processor as a backup processor to be used in the case of a failure.

The next step in this research is twofold, first to implement the prototype of this algorithm in our distributed and real-time operating system. We also plan to design a simulation model for this scheduling algorithm, the relationship between the rejection rate and the number of processors will be studied.

ACKNOWLEDGMENTS: The authors wish to thank Yanli Zhang for her helpful comments. This work was supported by National Defence Pre-research Foundation. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements.

VI. REFERENCES

[1] Ralf Steinmetz, "Analyzing the Multimedia Operating System" , IEEE Multimedia, Spring pp.68-84, 1995.

[2] Hiroyuki Kaneko, John A. Stankovic, "A multimedia Server on the Spring Real-time System", Compter Science Technical Report TR-96-11, University of Massachusetts, January 1996.
[3] John A. Stankovic, Krithi ramamritham, "The Spring kernel: A New Paradigm For Real-Time Systems", IEEE Software, pp.62-72 May 1991.
[4] R. Govindan and D.P. Anderson, "Scheduling and IPC Mechanisms for Continuous Media", In Proceedings of the 13th ACM Symposium on Operating Systems Principles. Association for Computing Machinery SIGOPS, pp.68-801991.
[5] Ramos Thuel S, and J.K. Strosnider, "The transient server approach to scheduling timecritical recovery operations", Real-time Systems Symposium, pp.286-295, 1991.
[6] Yingfeng Oh and Sang Son, "Fault-tolerant Real-Time Multiprocessor Scheduling", Technical Report TR-92-09, University of Virginia, April 1992.
[7] Yingfeng Oh and Sang Son, "Multiprocessor Support for Real-time Fault-tolerant Scheduling", In IEEE 1991 Workshop on Architectural Aspects of Real-Time Systems, San Antonio, TX, Dec 1991:76-80.
[8] A.L. Liestman and R.H. Campbell, "A Fault-tolerant Scheduling Problem", IEEE Trans. On Software Engineering, Vol.12, No.11, pp. 1089-1095 Nov. 1988.
[9] D.K. Pradhan. Fault Tolerant Computing: Theory and Techniques. JN: Prentice-Hall. 1986.
[10] B. Randell, "System Structure for Software Fault Tolerance", IEEE Trans. on Software Engineering, Vol.1, No.2, pp.220-232 Feb. 1975.
[11] Xiao Qin, L.P. Pang, S.L. Li and Z.F Han, "Efficient Scheduling Algorithm with Fault-tolerance for Real-time Tasks in Distributed Systems", In Proceedings of the 5th International Conference for Young Computer Scientists, ICYCS' 99, Nanjing, P.R. China, pp.721-725, Aug., 1999.
[12] Xiao Qin, Z.F Han, L.P. Pang and S.L. Li, "Static Real-Time Scheduling Algorithm with Fault-Tolerance in Multiprocessor Systems", In Proceedings of the 5th International Conference for Young Computer Scientists, ICYCS' 99, Nanjing, P.R. China, pp.765, Aug., 1999.
[13] Balaji S, "Workload Redistribution for Fault-tolerance in a Hard Real-time Distributed Computing Systems", Fault-tolerant Computer Systems19, Chicago, Illinois, pp.366-373 June, 1989.
[14] J. Chung, J. Liu and K. Lin, "Scheduling Periodic Tasks that Allow Imprecise Results", IEEE Trans. Computers, Vol.19, No.9, Sept. 1990.
[15] W. Shih, J. Liu and J. Chung, "Fast algorithms for Scheduling Imprecise Computations", Real-Time System Symposium, pp. 12-19 Dec. 1989.
[16] J. Liu, K. Lin, W. Shih, A. Yu, J. Chung and W. Zhao, "Algorithms for Scheduling Imprecise Computation", IEEE Computer, May, 1991.